Chase Moynihan chasemoy@csu.fullerton.edu
Mile McCloskey miles.mccloskey@csu.fullerton.edu
Diego Franchi diegofranchi@csu.fullerton.edu
CPSC 335 Project 3

Project 3 Report

Mathamatical Analysis:

------------------- Exhaustive Method --------------------

```
std::shared_ptr<Protein> exhaustive_best_match(ProteinVector & proteins, const
std::string & string1)
{
	int best_i = 0;                                          //O(1)
	int best_score = 0;                                      //O(1)
	for (int i = 0; i < proteins.size(); i++)                //O(n)
	{
 int score = exhaustive_longest_common_subsequence(proteins[i]->sequence, string1);//O(2(2^n(n))+n^2)
		if (score > best_score)                              //O(1)
		{
			best_score = score;                              //O(1)
			best_i = i;                                      //O(1)
		}
	}
	return proteins[best_i];                                 //O(1)

}
```

exhaustive_best_match Mathamatical Analysis:

$$=O(1 + 1 + n((2(2^n(n))+n^2)+1+1)+1)$$
$$=O(3 + n((2(2^n(n))+n^2))+2)$$
$$=O(n(2(2^n(n))+n^2))$$
$$=O(2n(2^n(n))+n^2)$$

Lemma: $(3 + n((2(2^n(n))+n^2))+2)$

$$\lim_{n\to\infty}\left(\frac{T(n)}{f(n)}\right) = \lim_{n\to\infty}\left(\frac{T(3+n((2(2^\wedge n(n))+n^\wedge 2))+2)}{f(2n(2^\wedge n(n))+n^\wedge 2)}\right) = 1$$

$$\therefore 3 + n((2(2^n(n))+n^2))+2 \in O((2n(2^n(n))+n^2))$$

```
int exhaustive_longest_common_subsequence(const std::string & string1,
	const std::string & string2)
{
	auto all_subseqs1 = generate_all_subsequences(string1);      //O(2^n(n))
	auto all_subseqs2 = generate_all_subsequences(string2);      //O(2^n(n))
	int best_score = 0;                                          //O(1)
	for (auto& s1 : *all_subseqs1)                               //O(n)
	{
		for (auto& s2 : *all_subseqs2)                           //O(n)
		{
			if (s1 == s2 && s1.length() > best_score)             //O(c)
				best_score = s1.length();                        //O(c)
		}
	}
	return best_score;
                                                                 //O(1)
}
```

Exhaustive_longest_common_subsequences Mathamatical Analysis:

$$= O((2^n(n))+(2^n(n))+1 + n(n(c+c))+1)$$

$$=O((2^n(n)) + (2^n(n)) _+ 2 + n(n(2c))$$
$$=O((2^n(n)) + (2^n(n)) + n^2)$$

Chase Moynihan chasemoy@csu.fullerton.edu
Mile McCloskey miles.mccloskey@csu.fullerton.edu
Diego Franchi diegofranchi@csu.fullerton.edu
CPSC 335 Project 3

$=O(2(2^n(n))+n^2)$

```
std::unique_ptr<std::vector<std::string>> generate_all_subsequences(const std::string
& sequence)
{
auto R = std::unique_ptr<std::vector<std::string>>(new std::vector<std::string>());
//O(c)
	double n = pow(2, sequence.length());                           //O(c)
	for (uint64_t bits = 0; bits < n; bits++)                       //O(2n)
	{
		std::string subsequence = "";                               //O(1)
		for (int j = 0; j < sequence.length(); j++)                 //O(n)
			if (((bits >> j) & 1) == 1)                             //O(c)
				subsequence += sequence[j];                         //O(c)
		R->push_back(subsequence);
                                                                    //O(1)
	}
	return R;
                                                                    //O(1)
}
```

generate_all_subsequences Mathamatical Analysis:

$=O(c + c + 2^n(1 + n(c + c) + 1)+1)$

$=O(2c + 1 + 2^n(2 + n(2c)))$

$=O(2c + 2^n(n(2c))$

$=O(2^n(n))$

------------------- Dynamic Programming ------------------

```
std::shared_ptr<Protein> dynamicprogramming_best_match(ProteinVector & proteins, const
std::string & string1)
{
	int best_i = 0;                                                 //O(1)
	int best_score = 0;                                             //O(1)
	for (int i = 0; i < proteins.size(); i++)                       //O(n)
	{
 int score = dynamicprogramming_longest_common_subsequence(proteins[i]->sequence, string1);//O(k + m + km2)

		if (score > best_score)                                     //O(1)
		{
			best_score = score;                                     //O(1)
			best_i = i;                                             //O(1)
		}
	}
	return proteins[best_i];
                                                                    //O(1)
}
```

dynamicprogramming_best_match Mathamatical Analysis:

$=O(1 + 1 + n((k + m + km^2)(1+1+1))+1)$

$=O(3 + n(k + m + km^2)(3))$

$=O(n(k + m + km^2)$

Lemma: $(3 + n(k + m + km^2)(3))$

$$\lim_{n\to\infty} \left(\frac{T(n)}{f(n)}\right) = \lim_{n\to\infty} \left(\frac{(3 + n(k + m + km^2)}{(n(k+m+km^2)}\right) = 1$$

$\therefore (3 + n(k + m + km^2)(3)) \in O(n(k + m + km^2))$

Chase Moynihan chasemoy@csu.fullerton.edu
Mile McCloskey miles.mccloskey@csu.fullerton.edu
Diego Franchi diegofranchi@csu.fullerton.edu
CPSC 335 Project 3

```
int dynamicprogramming_longest_common_subsequence(const std::string & string1,
      const std::string & string2)
{
       int n = string1.length();                                           //O(c)
       int m = string2.length();                                           //O(c)
       std::vector<std::vector<int>>D(n + 1, std::vector<int>(m + 1));//O(c)
       for (int i = 0; i < n; i++)                                         //O(k)
             D[i][0] = 0;                                                  //O(1)
      for (int j = 0; j < m; j++)                                          //O(m)
             D[0][j] = 0;                                                  //O(1)

      for (int i = 1; i <= n; i++)                                         //O(k)
      {
             for (int j = 1; j <= m; j++)                                  //O(m)
             {
                    int up = D[i - 1][j];                                  //O(c)
                    int left = D[i][j - 1];                                //O(C)
                    int diag = D[i - 1][j - 1];                            //O(C)
                    if (string1[i - 1] == string2[j - 1])                  //O(C)
                           diag++;                                         //O(1)
                    int intermediate = std::max(up, left);                 //O(C)
                    D[i][j] = std::max(intermediate, diag);                //O(C)
             }
      }
      return D[n][m];
                                                                       //O(1)
}
```

dynamicprogramming_longest_common_subsequence Mathamatical Analysis:

$=O(c+c+k(1)+ m(1) + k(m(c+c+c+c+1+c+c)+1)$

$=O(2c + k + m + km(6c+1)+1)$

$=O(2c + k + m + km(6c+km)+1)$

$=O(k + m + km(km))$

$=O(k + m + km^2)$

Chase Moynihan chasemoy@csu.fullerton.edu
Mile McCloskey miles.mccloskey@csu.fullerton.edu
Diego Franchi diegofranchi@csu.fullerton.edu
CPSC 335 Project 3

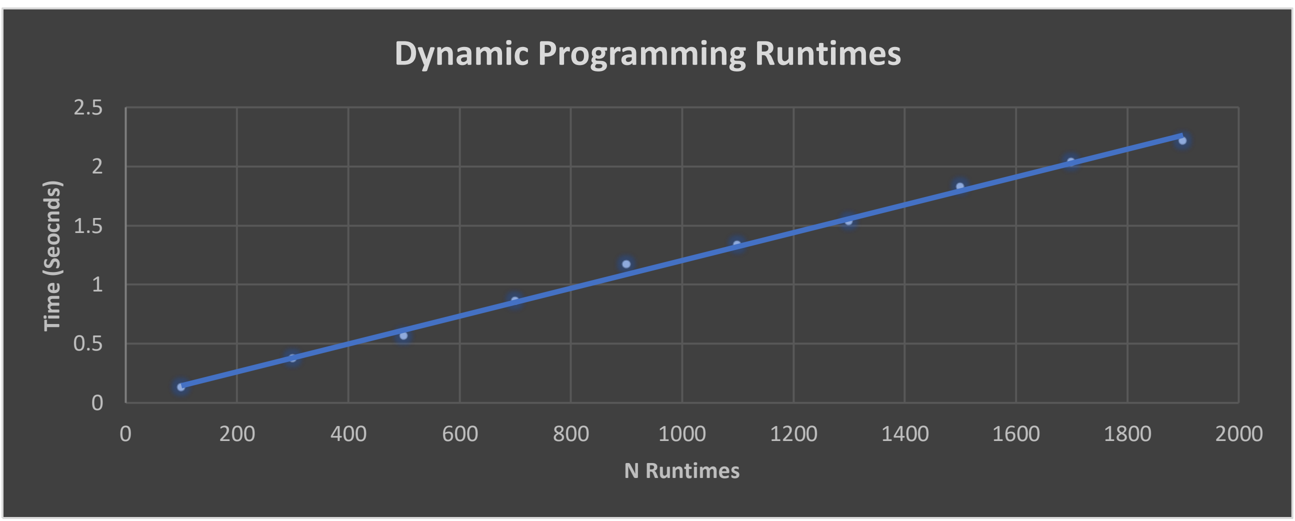
Empiracal Analysis:

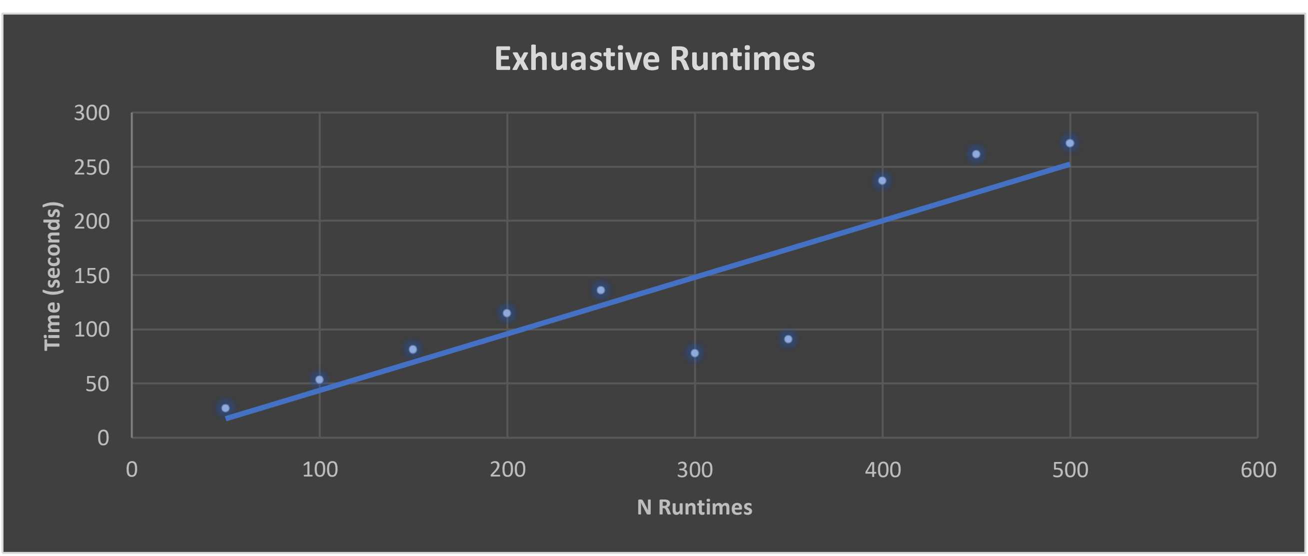
Dynamic Method:

n runtimes	time (seconds)
100	0.130544
300	0.375181
500	0.566103
700	0.860546
900	1.16766
1100	1.33538
1300	1.53265
1500	1.82574
1700	2.03512
1900	2.21351

Exhuastive Method:

n runtimes	time (seconds)
50	27.0331
100	53
150	81.3127
200	114.377
250	135.794
300	77.7037
350	90.6967
400	236.573
450	261.121
500	271.481





Chase Moynihan chasemoy@csu.fullerton.edu
Mile McCloskey miles.mccloskey@csu.fullerton.edu
Diego Franchi diegofranchi@csu.fullerton.edu
CPSC 335 Project 3

Conclusion

Exhuastive Match Results:

sp|P08521|AAP_YEAST

sp|Q02336|ADA2_YEAST

sp|P08521|AAP_YEAST

sp|P14164|ABF1_YEAST

sp|P19414|ACON_YEAST

Dynamic Match Results:

sp|P32469|DPH5_YEAST

sp|P32469|DPH5_YEAST

sp|P32469|DPH5_YEAST

sp|P32469|DPH5_YEAST

sp|Q08032|CDC45_YEAST

3)The results that we generated did not all match to the protein. Essentially, because its looking for two strings that matches, and both algorithms are looking for the longest subsequence, and when it finds them the two will ignore strings of equal length.

4)Our empirically-observed time efficiency data is consistent with the mathamatically derived big-O effiency class of the dynamic programming algorithm. However, our empirically-observed data isn’t as consistent with out mathamatically derived big-O efficiency class of the exhuastive algorithm.